Etape 2 : Interrompre un Thread JAVA

Multithreading en JAVA

Interruption : stopper certains traitements

- esprit de l'API JAVA : pas d'interruption arbitraire d'un thread depuis l'extérieur ce doit être une opération finalisée en interne au thread.
  - dans le cas général, un thread est insensible à une demande d'interruption de l'extérieur !
  - ce n'est pas une opération anodine : interrompre brutalement un traitement pourrait placer des ressources dans un état incohérent !

=> c'est presque toujours sous le contrôle du thread concerné !!

Interruption des *threads*

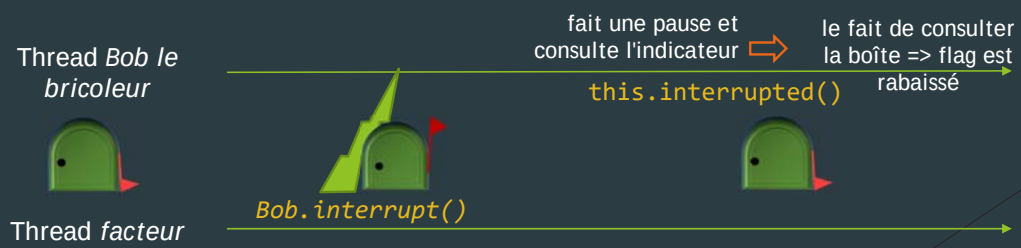
- demander l'interruption : méthode `interrupt()`
  - c'est une méthode d'instance qui vise à interrompre le thread sur lequel elle est appliquée
  - selon l'état courant du thread concerné (en cours d'exécution ou momentanément stoppé) les conséquences de l'invocation de la méthode ne sont pas les mêmes

- le type Thread dispose d'une variable indiquant si une instance a fait l'objet (ou pas) "d'une demande d'interruption"

Interruption des *threads*

- `isInterrupted()` => {True, False} indique si le thread sur lequel la méthode s'applique a reçu (True) ou pas (False) une demande d'interruption

- `interrupted()` => {True, False} indique si le thread sur lequel la méthode s'applique a reçu (True) ou pas (False) une demande d'interruption + avec cette méthode l'indicateur (le flag) est rabaissé après l'appel (il vaut toujours False)



Interrompre un thread

```
public class Exemple0 {

public static void main(String[] args) throws InterruptedException {

    TempsQuiPasse chronos = new TempsQuiPasse();

    chronos.start();

    System.err.println("[Main]: je vais interrompre le cours du temps");

    chronos.interrupt();

    System.err.println("[Main]: j'ai essayé d'interrompre chronos");
    }
}
```

Gérer une demande d'interruption

```
class TempsQuiPasse extends Thread {
    @Override
    public void run() {

        int i = 0;
        while (true) {
            System.out.println("[Chronos] Je fais passer le temps ... ");

            if (this.isInterrupted() && (i == 500)) {
                System.out.println("[Chronos] Je veux bien faire un break ...");
                break;
                }
            else
                System.out.println("[Chronos] Quelqu'un essaye vainement
            d'arrêter le temps ...");
            i++;
    }
}
```

Gérer une demande d'interruption

```
[Chronos] Je fais passer le temps ...
[Main]: je vais interrompre le cours du temps
[Main]: j'ai essayé d'interrompre chronos
[Chronos] Quelqu'un essaye vainement d'arrêter le temps ...
[Chronos] Je fais passer le temps ...
[Chronos] Quelqu'un essaye vainement d'arrêter le temps ...
[Chronos] Je fais passer le temps ...
[Chronos] Je veux bien faire un break ...
```

la fin des Thread

- par défaut les Thread Java ont une durée de vie complètement dépendante des traitements à réaliser dans leur méthode run.
  - => la JVM ne se termine qu'à la fin des Thread créés et démarrés dans son main
- On peut toutefois paramétrer un Thread de façon à ce qu'il soit arrêtés dès la fin du main : cas des démons
- On peut positionner cette propriété grâce à la méthode setDaemon() à invoquer impérativement avant le démarrage du thread

```
monThread.setDaemon(true);
```

  - à réserver a des Thread particuliers car les démons sont arrêtés brutalement par la JVM lorsqu'elle se termine